DOCUMENT RESUME

ED 233 914 SO 013 482

AUTHOR Veldman, Donald J.
TITLE Fundamental Fortran for Social Scientists.
INSTITUTION Texas Univ., Austin. Research and Development Center for Teacher Education.
REPORT NO RDCTE-R-4105
PUB DATE 80
NOTE 64p.
PUB TYPE Guides - General (050)

EDRS PRICE MF01/PC03 Plus Postage.
DESCRIPTORS Computer Programs; Computers; *Data Processing; *Programming; *Programing Languages; *Social Sciences; *Statistical Analysis
IDENTIFIERS *FORTRAN Programing Language

ABSTRACT

An introduction to Fortran programming specifically for social science statistical and routine data processing is provided. The first two sections of the manual describe the components of computer hardware and software. Topics include input, output, and mass storage devices; central memory; central processing unit; internal storage of data; and monitors, compilers, and Fortran programs. Section 3 describes the essentials of Fortran that are available in all compilers: the statement format, starting and stopping the program, and order of execution. Sections 4, 5, and 6 detail Fortran statements used to process data stored in the computer memory. These sections provide technical information on constants, variables, and arrays; expression and assignment statements; and control statements and loops. Section 7 explains statements that are used to transfer information between central memory and external media, such as punch cards, the printed page, and mass storage devices. Section 8 contains five sample exercises that begin with a simple problem which is gradually extended and complicated. Sections 9 and 10 describe subprograms and outline common problems in social science data processing. The manual assumes the characteristics of a CDC 6000 or Cyber series machine. Important differences between CDC and IBM Fortran are noted, however. A summary of Fortran statements concludes the document. (KC)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED233914

U.S. DEPARTMENT OF HEALTH, EDUCATION & WELFARE
NATIONAL INSTITUTE OF EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY.

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
O.H. Bown
TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

FUNDAMENTAL FORTRAN

FOR SOCIAL SCIENTISTS

Donald J. Veldman

Research and Development Center for Teacher Education

The University of Texas at Austin

1980

Report No. 4105

SO 013 482

CONTENTS

Preface. 2

I. Computer Hardware. 3

II. Computer Software. 10

III. The Fortran Language 15

IV. Constants, Variables, and Arrays 18

V. Expressions and Assignment Statements. 21

VI. Control Statements and Loops 25

VII. Input and Output 32

VIII. Example Programs 40

IX. Subroutines and Functions. 47

X. Some Useful Techniques 52

Appendix A: Differences Between CDC and IBM Fortran. 60

Appendix B: Summary of Fortran Statements 61

Preface

Many books on Fortran programming have been published, in addition to the reference manuals provided by computer manufacturers. These books have two serious drawbacks for beginning students of the language--particularly those in the social sciences. Almost all are oriented toward mathematics and engineering applications rather than statistics and routine data processing. Also, these books present all of the features of the language, swamping the student in a sea of details.

The purpose of this introductory manual is to provide the student with the bare essentials of the language--enough to accomplish almost any data processing task, but without the vast array of elaborations and minor conveniences that would be included in a full description of the language.

You will not find it difficult to master the material presented here. Once you have done so, you will discover that the hard part of computer programming is not putting a procedure into Fortran code; the hard part is specifying the procedure itself, as an unambiguous series of discrete steps. The only way to learn that is through practice. This manual will give you all the tools you really need. Once you are able to use the basic terms proficiently, you can go to the reference manuals to expand your vocabulary.

For the most part, a Fortran programmer needs to know little about the particular computer on which his programs will be run. This manual assumes the characteristics of a CDC 6000 or Cyber series machine. Users of IBM equipment will find an explanation of the important differences in Appendix A of this manual.

I. Computer Hardware

The essential parts of every computer system are shown in Figure 1. Although you need to know very little about the actual hardware configuration you use as a programmer, it does help to have some idea of the way these machines are organized.

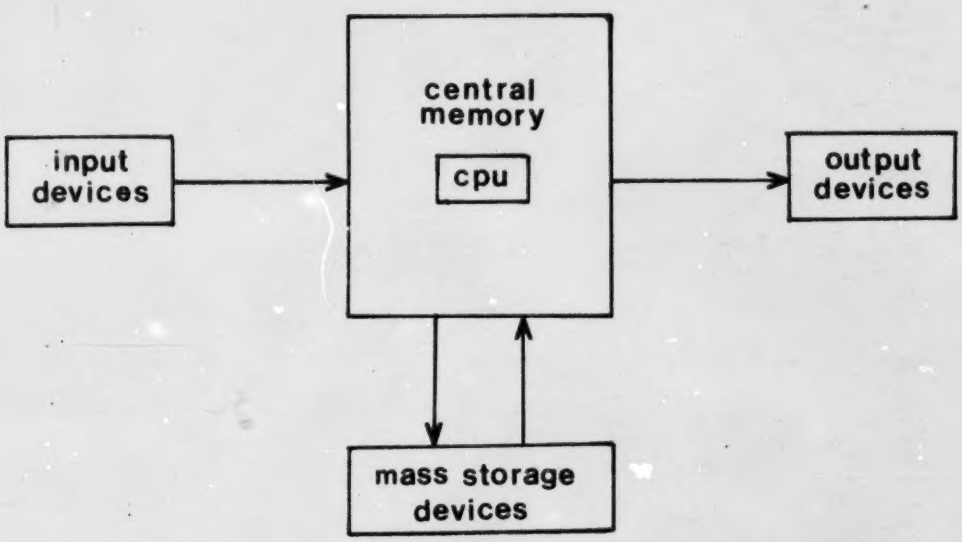


Figure 1. Major components of a computer system

Input Devices

The most common way of entering programs and data to a computer system is by means of punched cards. Most people these days are familiar with the standard 80-column card, in which each column is used to represent one character--a letter, numeral, or other symbol. Punch card codes are not entirely standardized; a widely used system is shown in Figure 2.

BEST COPY AVAILABLE

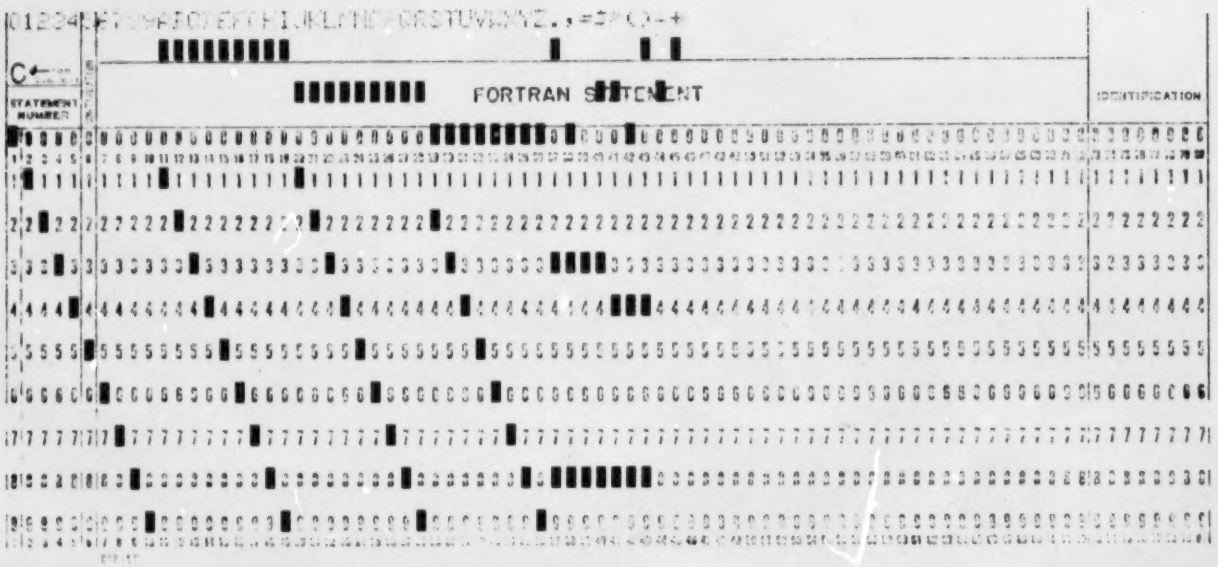


Figure 2. A commonly used punch card code

In addition to the card reader, information can be entered to a computer from a keyboard on a teletype machine or one connected to a TV screen terminal. Magnetic tape may also be used as the vehicle for entering data and even programs to the system. From the programmer's standpoint it makes little difference which method of data entry is employed.

Output Devices

Output as well as input to the computer may be in the form of punched cards or magnetic tape, but the most common medium is lines of print, either on paper or on the screen of an interactive terminal.

Punched cards or magnetic tape output from a program, of course, may serve as input to another program. From the programmer's viewpoint, as with input, there is little need to know which output device is used. That little will be explained later.

Central Memory

The fundamental unit of any computer memory is a binary device, such as a tiny iron doughnut or "core" through which wires are strung. This device can register one of two electronic conditions at any given instant: on or off, yes or no, 0 or 1--hence the term "binary." The unit of information represented is called a "bit"--short for "binary digit." Most computer memories contain millions of these devices.

The binary units of the memory are organized into "words." The number of binary units per word varies among computer models. We will assume a word size of 60 bits.

Each word of memory can hold a single quantitative value, or a set of 10 coded characters, or one or two instructions of a program. The power of modern computers is due to the fact that the program is stored in the same central memory that holds the data to be processed, thus permitting the program to modify itself during its operation, contingent on intermediate results.

Central Processing Unit (CPU)

This consists of a set of working registers and switching networks which execute the sequence of instructions in a program, obtaining and storing information in the central memory. At the level of machine language, instructions might be "Put the contents of the word at address 22056 into register A," or "Add the contents of registers A and B," or "Shift the bit string in register A four places to the left." Fortunately, the Fortran programmer does not have to be nearly so specific.

Access time to central memory and the time it takes to execute an instruction in a modern computer is measured in billionths of a second.

Because central memory is expensive, its storage capacity is limited and other devices with much longer access times are used to hold information waiting to be processed by the CPU.

Mass Storage Devices

There are two important types of these: tape drives and disk systems.

Tape. Computer tapes are either one-half or one inch wide, and are typically 2400 feet long. A tape can hold as much information as 256,000 cards. The advantage of tapes is that they can be demounted and stored when not in use by the computer.

Disk. These look roughly like stacks of phonograph records, on which information is stored in circular tracks. Most disk units do not permit demounting and storage. Average access time to information on disks is about two-thirds of that on tape.

The mass storage devices are controlled by the CPU, which uses them to hold input and output files waiting to be processed, and they may also be used during execution of a program for temporary storage of information.

Logical Unit Numbers

In a Fortran program, mass storage devices are referenced only as unit numbers. At the time the program is executed, these unit numbers are linked to specific pieces of equipment by means of "system control cards." The manner in which such device assignments are made varies from one to another computer installation.

Reading data from cards, punching of output data, and output to a line printer are implicit in the form of the Input/Output (I/O) statements used.

Information is transmitted by I/O state ents in units called "records." Maximum record sizes are 80 columns (characters) for punch cards and 136 characters for print lines and for records on a mass storage device.

Internal Storage of Data

Within the computer memory, information can be stored in a particular word in one of three ways. Which one is used is determined by the programmer, and the manner in which one does so will be discussed in Chapter VII. Figure 3 depicts the three ways in which the bit string of a word of memory can be employed.

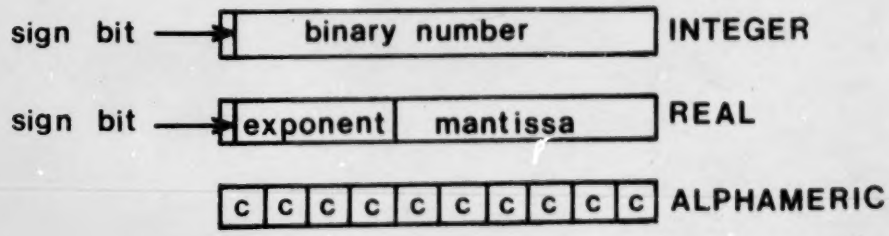


Figure 3. Three modes of data storage

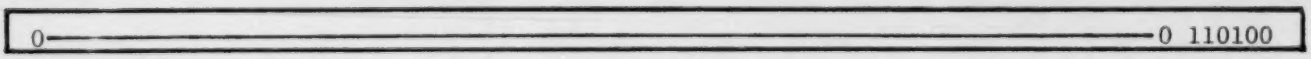
Integer Mode. A field (set of consecutive columns) on the input medium (e.g., card) is interpreted as a single numeric value and is translated directly from the decimal (base 10) to the binary (base 2) number system. The first bit in the word is used to indicate whether the number is positive or negative. Obviously, only whole numbers can be stored in this mode.

Real Mode. The input field is interpreted as a single numeric value. A pseudologarithmic notation is used to code the value as two binary numbers, one a mantissa and the other an exponent which locates the implicit decimal point. The first bit indicates whether the value is positive or

negative. Because the mantissa is shorter, precision is less than in integer mode, but any value, including fractions, can be represented.

Alphameric. In this mode the input field is not interpreted as a numeric value. Rather, each character in turn is converted to a unique binary number and stored in a set of six bits, beginning at the left end of the word of memory. Unused spaces are filled automatically with the code for a blank column. The maximum number of characters per word is ten (6 x 10 = 60).

An Example. Suppose we read two columns of a card that contains the symbols "52" into memory. If we read them as integer mode data, the machine will interpret the field as the number fifty-two, and translate 52 (base 10) to its equivalent in the binary number system (base 2), as shown below.



INTEGER MODE

$52_{10} = 110100_2$

If, however, we read the same two card columns as a real mode value, it will be stored as a mantissa and an exponent to locate the implicit decimal point.

exponent **mantissa**

REAL MODE

The third possibility is to read the two columns alphamerically, not as a quantitative value, but simply as two characters.

100000	011101	101101	101101	101101	101101	101101	101101	101101	101101

ALPHAMERIC MODE

"5" = 100000_2

"2" = 011101_2

"Δ" = 101101_2

II. Computer Software

The term "software" means the programs of instructions that control the activity of the hardware, the machinery. There are three types of programs you need to know about: monitors, compilers, and Fortran programs. It is the third type that you will learn to write.

The Monitor

This program is always in the computer system and always running. It keeps the accounts, assigns priorities to jobs entering the system, and generally keeps track of what is going on. Communication with the monitor is by means of a "job control language" that varies from one computer to another. Your instructor or someone at your computation center will tell you what you need to know about the system control cards that are necessary to run a program.

The Compiler

The compiler is a program that translates statements written in the Fortran language into the specific machine language instructions that the CPU can understand and execute. There is often more than one Fortran compiler available at a particular installation. Compilers vary in speed of translation, running speed of the object code they produce, and the extensiveness of error diagnostics they provide. The compiler is always stored in the computer system and is called into operation with a particular system control card.

Sequence of Processing

The input file (e.g.,card deck) that the programmer submits to the computer system consists of three parts:

(1) System control cards, which are processed by the monitor

(2) The Fortran program (and subroutines), which are processed by the compiler

(3) The data to be processed by the compiled program.

The sequence of operations carried out by the computer system when it processes this file is as follows:

1. The system control cards are read, priority for the job is assigned, necessary tapes are mounted, and the Fortran compiler is obtained from disk.
2. Control is turned over to the Fortran compiler program which translates the source program into a machine language object program, flagging any syntax errors it discovers in the process. If errors are found, the job is usually aborted.
3. Control is turned over to the object program by the monitor, and it processes the data. Errors may also occur during this processing which would cause the job to be aborted. For instance, attempted division by zero is a fatal error in most systems.

Program Structure

We will discuss this topic here in very general terms, and return to it later with specific examples. The simplest program structure involves three major operations:

- Input all data to memory
- Compute all results
- Output all results from memory

Quite often the problem involves separate processing of data from a series of sources (e.g., subjects). Far less memory may be needed if we structure the program differently:

∘For each case in turn:

∘∘Input data to memory

∘∘Compute results

∘∘Output results from memory

A common structure in statistical analysis involves partial results, such as the accumulation of sums and sums of squares.

∘For each case in turn:

∘∘Input data to memory

∘∘Compute partial results

∘Compute final results

∘Output final results from memory

As a final example, the problem may involve a series of variables to be independently processed, and too many partial results to store them all in memory at once. Temporary storage of the data on tape or disk is necessary because a program can only read a data deck once.

∘Rewind tape

∘For each case in turn:

∘∘Input data to memory

∘∘Output data to tape

∘For each variable in turn:

∘∘Rewind tape

∘∘For each case in turn:

∘∘∘Input data from tape

∘∘∘Compute partial results

∘∘Compute final results

∘∘Output final results from memory

Subroutines and Functions

These are independently compiled Fortran routines that can be called into operation by single statements in programs or other subprograms. They are linked to the calling routines by a list of arguments. Consider the formula for the correlation coefficient (r_{xy}) which employs means (μ_x, μ_y) and sigmas (σ_x, σ_y).

$$r_{xy} = \frac{\Sigma XY/N - \mu_x \mu_y}{\sigma_x \sigma_y}$$

$$\mu_A = \Sigma A/N \qquad \sigma_A = \sqrt{\Sigma A^2/N - \mu_A^2}$$

These formulas employ a nested structure of computations, as shown in Figure 4. The computation of each μ and σ could be accomplished by subroutines or functions. In fact, the computation of r could be carried out in a subroutine, which would itself be called by a master program. The sequence of operations in the r_{xy} program would be as follows:

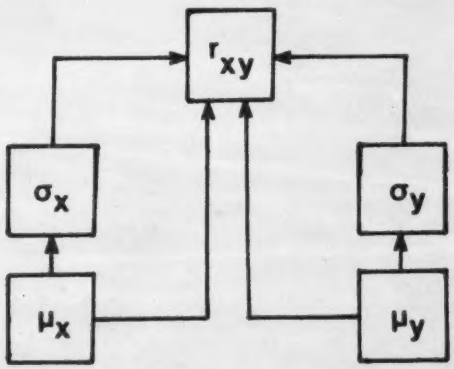


Figure 4. Nested formulas for a correlation coefficient

1. Accumulate ΣX, ΣY, ΣX^2, ΣY^2, ΣXY from N cases
2. Define μ_x by calling function A (arguments ΣX and N)
3. Define μ_y by calling function A (arguments ΣY and N)
4. Define σ_x by calling function S (arguments ΣX^2, μ_x, and N)
5. Define σ_y by calling function S (arguments ΣY^2, μ_y, and N)
6. Compute r_{xy}
7. Output μ_x, μ_y, σ_x, σ_y, and r_{xy}

Subroutines and functions are useful in three situations:

(1) When a sequence of operations is needed at different places in a program (e.g., computing a sigma).

(2) When a sequence of operations is needed in different programs (e.g., computing a matrix of intercorrelations).

(3) When a long complex procedure can be blocked into a more manageable series of major processes.

III. The Fortran Language

Fortran was created in 1954 by a group of 13 men at International Business Machines. The name of the language comes from FORmula TRANslation, since its original purpose was to allow scientists to state mathematical problems for machine solution. It is by far the most commonly used language today.

Many different "dialects" of the language exist, with more or less restricted vocabularies and grammar, depending primarily on the size of the computer system. We will not be concerned with such differences here, since we will present only essentials of the language that are available in all compilers.

Statement Format

The basic unit of a Fortran program is the statement, which normally occupies one punch card, but which can be continued on subsequent cards if necessary. The statement card has a particular format which is illustrated in Figure 2 (Chapter I).

Column 1. A card containing the letter "C" in column one will be considered a comment card and not a part of the program. Otherwise, this column is left blank.

Columns 2-5. In this field the statement may be assigned a statement number. Only certain statements--those referenced by other statements--need to be numbered. The numbers may be assigned without regard to quantitative value, since the numbers serve only as labels. In particular, the values of statement numbers do not determine the order of execution of the statements. The numbers must be positive integers assigned uniquely to single statements.

Column 6. In one-card statements this column is left blank. If a continuation mark (use serial number 1, 2, 3, etc.) appears here, the card will be considered a continuation of the previous card of the statement; therefore, this column must be blank in the first card of a multiple-card statement.

Column 7-72. This field contains the text of the Fortran statement. With one exception to be explained later, spacing within the text field is of no importance.

Column 73-80. This field is ignored by the compiler, and may be used to give the cards sequence numbers, which are helpful should you drop the deck accidentally.

The field format described here applies only to cards holding statements in a Fortran program. It has nothing to do with data cards, in which you can format all 80 columns in any way you wish.

Starting and Stopping the Program

The first card in every program begins in column seven with the word "PROGRAM" followed by an arbitrary name. This name may be up to six characters long and must begin with an alphabetic character. Following this name, a series of "buffer names" appears in parentheses.

If cards are to be read by the program, the buffer name INPUT is included. If any printed output will be produced, OUTPUT is needed. Punch card output requires PUNCH. Mass storage units (disk and/or tape) to be used are each designated by the word TAPE followed by the unit number employed in the program.

Example: PROGRAM VULCAN (INPUT, OUTPUT, TAPE 2)

The last card in every Fortran program (and subroutine and function, for that matter) contains the word "END" beginning in column seven. This serves as a signal to the compiler that the end of an independent routine has been reached. The END statement must not be numbered, since it is not an operational part of the program.

The word STOP is used to indicate that execution of the program is to cease, returning control of the job to the monitor. Usually the STOP statement will appear just before END, but STOP statements may be used elsewhere, such as when an error condition is encountered.

Order of Execution

Unless specifically directed otherwise by one of the three control statements described in Chapter VI, statements are executed in the order that they appear in the program.

Not all statements are operational; some are used only as reference information by other statements. One statement in particular, the word "CONTINUE," serves only as a reference point, and is always given a statement number.

Fortran programming is somewhat like giving instructions to a very literal-minded, but extremely efficient, clerk--one with a limited vocabulary, as far as this manual is concerned. If the instructions you give are wrong, or are given in the wrong order, the clerk will produce nonsense--with blinding speed. It is very important to recognize that successful execution of a program is no guarantee that the answers are correct.

Random machine errors are extremely rare in modern computers. If a program fails, go to your instructor or an experienced programmer to find out why. "Magical thinking" and guessing will only handicap you in learning to program.

IV. Constants, Variables, and Arrays

In Chapter I we described three ways in which data can be represented in a word of computer memory. In this chapter we will discuss the way these memory cells are referenced in a Fortran program. Generally, this is done by giving names to the cells; the compiler assigns these names to particular memory addresses.

Constants

A constant is a particular numeric value or a string of alphameric characters. The first time the compiler encounters a constant in a program, it assigns it to a specific memory cell and stores it there. It remains unchanged during execution of the program. Examples of the three kinds of constants follow.

Integer. No decimal point appears: 2 -23 106

Real. Decimal point used: 0.5 -32.0 233.7504

Alphameric. The letter "H" is preceded by the length of the constant in characters, and the character string follows the H. Maximum length is ten characters: 4HMEAN 1HA 5HSIGMA

Variables

Variable names are assigned to specific memory cells by the compiler. The contents of these cells may be changed repeatedly during program execution. Their contents are indeterminate until defined by instructions in the program.

A variable name may be up to six characters in length and must begin with an alphabetic character. Variables holding numeric values are considered to be one of two modes, depending on the first letter of the name. A variable is considered to be integer mode if the first letter of

its name is I, J, K, L, M, or N. If the first letter is any other, the variable is considered to be real mode. If the variable is to be used for storage of alphameric data, either mode name may be used, but should be used consistently for all such variables in the program, for reasons to be explained later. Examples follow:

Integer: K MEAN J2

Real: AVG X DATA33

Arrays

A variable name may be used to refer to a series of memory locations, called an array. An array may have one dimension (a list), two dimensions (row by column table) or three dimensions (rows, columns, and slices).

To be used as an array variable, the name must appear in a special reference statement at the beginning of the program. Following the word DIMENSION, the names of all array variables are listed; following each name are the array dimensions in parentheses. Order of the variables in the list does not matter. An example follows:

```
DIMENSION A(3,10), SIG(4), KX(2,7,4)
```

Three array variables are to be used in the program: A has three rows and 10 columns, SIG is a list of four locations, and KX has a total of 56 elements arranged in two rows, seven columns and four slices.

Any variable name that does not appear in the DIMENSION statement will be considered a simple or one-location variable. Almost every useful program you will write will require a DIMENSION statement.

Subscripts

With only two exceptions to be described later, every appearance of an array variable name is followed immediately by the appropriate number

of subscripts in parentheses. Subscripts may be positive integer constants, simple integer variables, or simple integer expressions. Examples follow:

A(2,3) SIG(KA) KX(I*2,K,J-4)

Commas

The placement of commas in Fortran statements is very important. In the example statements here and to follow, note carefully where they appear and do not appear. For instance, multiple dimensions of arrays are separated by commas, as are multiple subscripts, as well as specifications of multiple arrays, but no comma appears after the word DIMENSION.

V. Expressions and Assignment Statements

The term expression refers to a group of variables and/or constants connected by arithmetic operators. The five arithmetic operation symbols are:

+ addition

- subtraction

* multiplication

/ division

** exponentiation

Unlike algebraic notation, variable names and constants must be separated by operators. Because it squeezes out blanks, the Fortran compiler would treat A B as the variable name AB. Multiplication of A by B would have to be written A*B.

Neither may two operators appear together without intervening variables or constants, unless parentheses separate them. For instance, A*(-B) is acceptable.

As an example, consider the following formula which defines sigma, the parameter standard deviation:

$$\sigma_x = \sqrt{\frac{\Sigma X^2}{N} - \mu^2}$$

An equivalent Fortran expression is:

```
(X2 / N - A * A) **0.5
```

Before this expression could be evaluated, the sum of squares would have to be accumulated and stored in X2, and the mean computed and stored in A. N would also have to be given an appropriate value. Note that exponentiation by 0.5 is the same as taking a square root, and that A*A could be written A**2.

Order of Evaluation

Evaluating an expression means reducing it to a single value by carrying out the operations specified. A particular order will be followed, unless parentheses are used to force a different order. Operations within parentheses are carried out first. Within parentheses, if any, exponentiations are done first, then multiplication and division, and then addition and subtraction.

In the previous example, this would be the order of evaluation:

1. Divide X2 by N
2. Multiply A by itself
3. Subtract the A*A value from X2/N
4. Exponentiate the result by 0.5

Assignment Statements

The equals sign identifies the basic operational Fortran statement, which changes the contents of a single memory cell. The simplest form of this statement transfers the content of one location to another.

```
A = B
```

The current content of A is replaced by that of B. Location B is not affected. If A was an array, the appropriate subscript(s) would have to be added. The same is true of B.

An expression may appear on the right side of the equals sign:

```
S = (X2 / N - A * A) **0.5
```

In all cases, no variables other then the one named on the left side of the equals sign will be affected.

The equals sign, then, is a transfer operator that means "replace the content of the location named on the left with the result of the operations specified on the right."

Accumulation of a series of values is accomplished by repeatedly executing a statement such as this, stepping the value of K each time:

```
A = A + X(K)
```

"Replace the current value of A with the sum of the current value and that in the Kth location of array X."

The manner in which the stepping of K is most easily accomplished will be discussed in the next chapter.

Mode Conversion

If the modes (integer and real) of the variables and constants in an expression are not all the same, integer values will be changed to real mode before the operations are carried out. Note, however, that in the following example, all values in the parentheses are integer mode, and the division will be done in that mode. The result of this will then be changed to real mode before the addition.

```
X = (K / 5) + A
```

The reason that this is important is that when the result of integer division is not a whole number, the fractional remainder is lost. If K had held the value 19, the result of the division would have been the value 3. This is called truncation.

If the modes are different on the two sides of an equals sign, automatic conversion to the mode on the left will take place just before storage. Again, if the conversion is from real to integer, any decimal fraction will be truncated, not rounded.

Test Question

Work through the following series of statements to see whether you understand what we have presented so far.

```
DIMENSION X(3)
X(1) = 5.0
X(2) = 2.7
X(3) = 7.0
K = X(2) / 3.0
R = X(1) + X(3) * K
```

If you did not come up with a final value in R of 5.0, perhaps you did not realize the K will be zero, or that multiplication precedes addition.

Exercises

```
DIMENSION A(2,3), M(5)
```

A

2.3	1.0	-10.5
5.6	0.0	2.1

M

3	-2	0	2	-1

1. `K = A(1,1) + A(2,3) / 2`

2. `X = (A(1,3) + A(2,1)/3) - M(2) * M(4)`

3. `X = ((A(2,3)**2) * A(2,2)) + (A(1,2) - A(1,1))`

   Remove any unnecessary parentheses and evaluate.

4. ```
   K = M(2)**2 - M(1)
   L = M(3) / 3 + M(4) / 2
   X = A(K,L) + A(L,K) / 23
   ```

5. Write the following as a Fortran statement:

$$P = \frac{A^2 B^2}{X-2} + \sqrt{\frac{(X-1)^2}{(X+1)}}$$
   ```

VI. Control Statements and Loops

In this chapter we will describe three statements that permit the programmer to order the execution of statements other than as they appear in the program. All three of these control statements reference some other statement by number.

Unconditional Transfer

The GO TO statement transfers control to the statement whose number follows the words GO TO. That statement will be executed next and control will continue from there.

Conditional Transfer

The word IF is followed by a proposition in parentheses. This proposition is evaluated as being either true or false. If the proposition is true, the Fortran statement following the parentheses is executed. If the proposition is false, this Fortran statement is skipped and control continues to the statement following the IF. The statement following the parentheses may not be numbered, may not be a DO statement (discussed later), and may not be another IF. The general form, therefore, is:

IF (proposition) statement

Relational and logical operators are used to state the proposition to be evaluated for truth or falsity. The six relational operators are:

.LT. less than

.LE. less than or equal to

.EQ. equal to

.NE. not equal to

.GE. greater than or equal to

.GT. greater than

For example, the following proposition is true if the current value in X is 5.0, and the GO TO statement would be executed.

```
IF (X .LE. 7.0) GO TO 4
```

Arithmetic expressions may also be used:

```
IF (DX + 42.5 .NE. A / 6.0) L = L + 1
```

The arithmetic expressions would be evaluated before the relation would be evaluated.

Relational propositions may also be linked by the logical operators .AND. and .OR. With .AND. both propositions must be true, while with .OR. either one may be true, to lead to execution of the statement on the IF line. In the following example, the final evaluation would be true if K was 5 and J was 3.

```
IF (K .LT. 6 .AND. J .EQ. 3) D = 45.0
```

Loops

Consider the following code, which would accumulate in A the sum of the five values in X.

```
      A = 0.0
      K = 1
    2 A = A + X(K)
      K = K + 1
      IF (K .LE. 5) GO TO 2
```

This is called a loop. After the fifth value from array X has been added to A, K will be raised to six, and the proposition in parentheses will be false, so the GO TO statement will not be executed, and control will continue on to the statement under the IF line.

It is necessary to zero the accumulator cell A before beginning the

loop, since the value in it is indeterminate unless explicitly set. K also must be initialized as one.

If you are not completely sure how this works, draw boxes for A, K, and the five cells of X, and step through the code, changing the values in the cells as dictated by the statements as they are executed.

The DO Statement

This statement allows economical specification of loops. The initialization of the index variable, the stepping of it, and the return decision are all implicit in the DO statement. For example:

```
DO 22 I = 1,40,2
```

This implies the following: "Do each of the statements in turn from the one following this, through statement 22, using index variable I to keep track of the passes, beginning with I equal to one, through I equal to 40, stepping I by two after each pass."

When the increment (step value) is one, the third parameter may be omitted. The index variable may be any integer variable name and the three parameters may also be integer variables. None of these variables may be changed by statements in the loop, however.

In the example above, control would continue on to the statement following number 22 after the last pass--when I was 39. Note that I will take on only odd-number values, and never will reach 40.

Here is the simplest possible DO loop, which zeros out ten locations in array X.

```
    DO 2 I = 1,10
  2 X(I) = 0.0
```

The accumulation loop described earlier could have been specified with three statements instead of five:

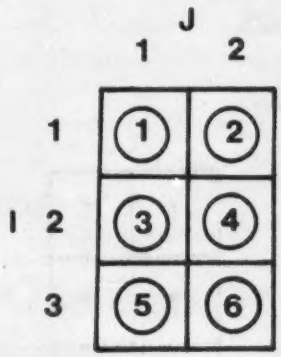
```
      A = 0.0
      DO 2 K = 1,5
    2 A = A + X(K)
```

Nested Loops

The series of statements from the DO statement to the numbered statement it specifies is called the range of the loop. The range of one loop may lie completely within the range of another, or the two loops may terminate on the same statement. When two loops are nested, the index of the inner (shorter) loop will be stepped through all of its values before the outer loop index is incremented.

Suppose we wish to zero all cells of matrix A. The following sequence will do the job, and the diagram indicates the order in which the cells will be zeroed.

```
      DO 2 I = 1,3
      DO 2 J = 1,2
    2 A(I,J) = 0.0
```



Care is necessary to coordinate the DIMENSION statement, the DO variables and limits, and the subscript references to them.

It is legitimate to transfer control (with an IF or GO TO) out of the range of a loop before it has been completed, and the current value of the index variable will be retained. When a loop has been completed, however, the index variable cannot be assumed to retain its final value.

Control must never be transfered into the range of a DO loop except through the DO statement itself.

DO loops may not terminate on another DO statement nor on a GO TO statement. Not all compilers will properly handle DO loops that terminate on an IF statement; a CONTINUE should be used instead:

```
      DO 2 I = 1,10
      IF (X(I) .EQ. 0.0) X(I) = 50.0
    2 CONTINUE
```

A few cautions are necessary concerning the parameters of a DO statement when variables are used for them: They must not inadvertently ever be zero or negative values. A DO loop will always be executed once, even if the first parameter is larger than the second.

Summation Code

Following are some summation formulae commonly used in statistics, along with equivalent Fortran code.

$$T = \sum_{1}^{N} X_i$$

```
      T = 0
      DO 2 I = 1,N
    2 T = T + X(I)
```

$$T = \sum_{1}^{N} X_i^2$$

```
      T = 0
      DO 2 I = 1,N
    2 T = T + X(I)**2
```

$$T = \left(\sum_{1}^{N} X_i\right)^2$$

```
      T = 0
      DO 2 I = 1,N
    2 T = T + X(I)
      T = T**2
```

$$T = \sum_{1}^{MN} \sum X_{ij}$$

```
      T = 0
      DO 2 I = 1,M
      DO 2 J = 1,N
    2 T = T + X(I,J)
```

$$T = \sum_{1}^{N} \left(\sum_{1}^{M} X_{ij} \right)^2$$

```
      T = 0
      DO 4 I = 1,N
      G = 0
      DO 2 J = 1,M
    2 G = G + X(I,J)
    4 T = T + G**2
```

Test Question

Step through the code for the last example above with the data below. If you get a final value for T other than 77, you don't understand DO loops yet (or you need practice in arithmetic).

N=3
M=2

X

I \ J		
	2	4
	3	1
	2	3

Exercises

DIMENSION A(2,3), B(3,2), M(6)

1. Write code to store the transpose of matrix A into matrix B.
2. Write code to store the largest (absolute) value in matrix A into a variable named X.

3. Write code to store row two of A followed by row one into M.

4. Determine whether or not B is the transpose of A and zero all cells of M if it is.

5. A and B hold integers between one and six. Put counts of each into cells of M.

VII. Input and Output

So far, we have described Fortran statements that are used to process data stored in the computer memory. In this chapter we will explain the statements that are used to transfer information between central memory and external media, such as punch cards, the printed page, and mass storage devices.

All of the input/output (I/O) statements to be discussed include the number of a reference statement (FORMAT) that describes the arrangement of fields of the records on the external medium. The maximum length of a record (in characters) is a function of the medium involved; punch cards are 80 columns long, while all other records, including print lines, have a maximum length of 136 characters.

READ, PRINT, and PUNCH

The general form of these three statements is shown below.

```
READ f, variable list
PRINT f, variable list
PUNCH f, variable list
```

After the initial word is the statement number of a FORMAT statement (f). The amount and order of the input or output is determined solely by the list of variable names in the I/O statement; the FORMAT statement serves only as a reference. Because FORMAT is not operational, FORMAT statements must not be referenced by the control statements GO TO, IF, and DO.

READ is used to enter data to the memory from punch cards; PRINT produces output on a line printer; and PUNCH yields output of punch cards.

Mass Storage I/O

The records on an external medium (tape or disk) together constitute a file. Before input or output operations begin with a particular unit, it must be explicitly positioned at the beginning of the file with the following statement, which specifies the unit concerned.

```
REWIND 2
```

There are two statements for manipulation of mass storage devices. Each includes in parentheses the number of the unit concerned (u) and the number of a FORMAT statement (f) that describes the fields of one or more records.

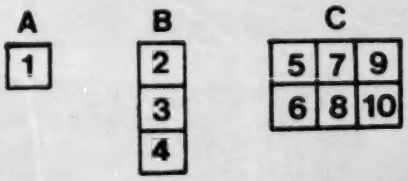
```
READ (u,f) variable list
WRITE (u,f) variable list
```

Unit numbers are arbitrarily chosen by the programmer, and are linked to particular pieces of equipment--tape or disk drives--by means of system control cards when the program is run.

Variable Lists

I/O statements are one of the two exceptions to the rule that an array name must be followed by appropriate subscripts. When an array name appears in an I/O variable list without subscripts, all elements of the array will be automatically transferred in a predetermined order. An example is shown below.

```
DIMENSION B(3), C(2,3)
READ 4, A, B, C
```



The simple variable A receives the first value, then all cells of B are filled, and finally data are entered to all elements of C. The natural array order for a matrix is down columns successively. In terms of changing subscripts, think of them as the reverse of a digital clock; the first (rows) varies most rapidly, then the second (columns), and the third (slices) changes most slowly.

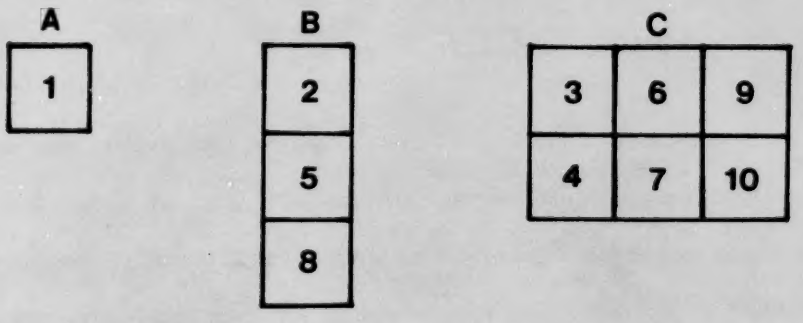
When the order of entry needed for a matrix is other than the natural one, or when an array is to be partially filled, an indexing notation like that of a DO statement is used. Parentheses enclose each loop.

```
READ 3, (B(I), I = 1,3)
```

As with the DO statement, a third parameter for a step value other than one may be used also. The example above does the same thing as the name B alone in the list. To enter values to C across rows instead of down columns, nested indexing would be necessary. As with nested DO loops, the innermost parenthesized index goes through all its values before the outer one is incremented.

```
READ 6, ((C(I,J), J = 1,3), I = 1,2)
```

One last illustration: Suppose we want to enter data to A, B, and C in this particular order:



This would be the necessary indexing:

```
READ 4, A, (B(J), (C(I,J), I = 1,2), J = 1,3)
```

FORMAT Statements

All FORMAT statements are numbered and are referenced only by I/O statements. For this reason they may be placed anywhere in the program deck that is convenient--usually after the I/O statement that first references them. The same FORMAT may be referenced by more than one I/O statement.

After the word FORMAT, a series of field specifications, enclosed by parentheses, designates the arrangement of one or more records on the external medium. Four particular letter codes are used for types of fields. Three of these codes (I, F, and A) correspond to the three ways that data may be represented in words of the memory, as described in Chapter I.

X fields. The number before the letter X indicates how many consecutive columns are to be skipped. Example: 5X

I fields. The number after the letter I indicates the length of the integer mode field in columns. A number in front of the I can be used to designate repetition of the field. Example: 3I2

F fields. The number immediately after the letter F indicates the total length of the real mode field in columns. This is followed by a period and then the number of decimal places at the right end of the field. In input fields the decimal point need not be explicitly present, but a column must be reserved for it in output fields. A repetition factor may be used before the letter F. On output, real mode values will be rounded to the number of decimal places specified. Example: 2F7.3

A fields. The length of the alphameric mode field in columns (characters) follows the letter A, and a repetition factor may precede it. The maximum length of such a field is ten characters. Example: 4A6

In both I and F fields, separate columns are required for minus signs, which may not be preceded in the fields by anything but blanks. Plus signs are never used for input or output. An example involving all four codes is shown below.

```
      DIMENSION KB(3), X(2)
      READ 5, ID, KB, X
    5 FORMAT (A4, 2X, 3I2, 3X, 2F5.2)
```

The FORMAT designates only as many columns of the record as necessary, but begins with column one. In the example above, an alphameric identification field is picked up from columns 1-4 and stored in ID. Columns 5-6 are then skipped. Three integer values, each two columns long, are then obtained from columns 7-12, and columns 13-15 are skipped. Finally two real-mode fields are entered from columns 16-25. Each of these is five columns long, the last two of which represent the decimal fraction.

The correspondence of variable name modes (integer and real) with FORMAT specifications (I and F) is crucial, and is not checked by the compiler.

Multiple Record Formats

More than one record may be described in a FORMAT statement; a slash mark (/) is used to tell the machine to end the current record and begin a new one. Specifications after the slash describe the new record beginning in column one. A slash at the beginning of a FORMAT will generate an empty record (double spacing of print lines).

If the I/O statement's variable list is not satisfied (completed) when the end of the FORMAT is reached, the entire FORMAT is repeated until all values have been transferred. Consider the following example.

```
      DIMENSION X(10)
      PRINT 6, X
    6 FORMAT ( / 1X, 3F5.1 / 1X, 3F5.1)
```

The following output would be produced. The delta symbol represents a blank character.

```
(blank line)
ΔΔ1.0ΔΔ2.0ΔΔ3.0
ΔΔ4.0ΔΔ5.0ΔΔ6.0
(blank line)
ΔΔ7.0ΔΔ8.0ΔΔ9.0
Δ10.0
```

Printer Control

Careful inspection of the previous example shows that each line is missing an initial blank character. The reason is that the first character in a record will not be printed, but will be used to control the printer. Column one in FORMATs to be used for printed output should be explicitly skipped, or implicitly left blank by making the first field longer than necessary. The only exception is when you want a particular print line to appear at the top of a new page, in which case the number one is used as the first character in the record. Other printer control characters exist, but vary among computer systems.

Headings

Strings of alphabetic characters may be inserted into output records by using the same nH method that is used to store alphameric constants in

memory. In FORMAT statements, however, such strings may be of any length, limited only by maximum record size. An example follows.

```
      PRINT 2, S
    2 FORMAT (21HΔSTANDARDΔDEVIATIONΔ=, F10.4)
```

An I/O statement need not always have a variable list. For instance, if all you want to do is generate a line at the top of a new page with a label on it, this will do it:

```
      PRINT 7
    7 FORMAT (6H1MEANS)
```

Subformat Repetition

A section of a FORMAT statement can be automatically repeated by using parentheses to enclose the section concerned, with a repetition factor preceding it.

```
      DIMENSION X(2,3)
      PRINT 8, X
    8 FORMAT (10H1MEANΔΔSIG, 3( / F5.2, F5.3))
```

X

2.30	2.45	2.15
.015	.200	.165

If the contents of X were as shown above, the print lines produced would be as follows, after a page eject.

```
MEANΔΔSIG
2.30Δ.015
2.45Δ.200
2.15Δ.165
```

If the slash in this example had been omitted, only one record (line) would have been produced.

Test Question

A deck of 5 cards, each containing two three-digit fields with one decimal place each in columns 6-11, is to be read into array X in the order shown below. Write the necessary READ and FORMAT statements.

X

1	2
3	4
5	6
7	8
9	10

Answer:

READ 2, ((X(I,J),J=1,2),I=1,5)
2 FORMAT
(5X, 2F3.1)

Exercises

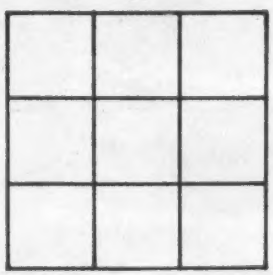
1. Find four errors:

```
PRINT 14, X, Y, Z, K
FORMAT ( / F1.1, F2.0, 2I3)
```

2. Indicate output order:

```
PUNCH 2, ((A(K,J), J = 1,3), K = 2,3)
2 FORMAT ( / 2F5.0 / 3F4.0)
```

A



3. How many cards will be output by the statements above?
4. Write code to transfer ID and 20 scores from card to tape for 1000 subjects (A8, 2X, 20F2.0)
5. Find four errors:

```
DIMENSION A(100,30)
DO 2 I = 1,100
REWIND 2
2 WRITE (3,2) A
3 FORMAT (30F5.2)
```

VIII. Example Programs

In this chapter we will begin with a very simple problem, and then gradually extend and complicate it.

Program ONE. Compute a Mean

Suppose we have a sample of 25 scores, each punched on a separate card in columns 6-10. We want to compute the mean of these values.

```
      PROGRAM ONE (INPUT, OUTPUT)
      A = 0.0
      DO 2 I = 1,25
      READ 1, X
    1 FORMAT (5X, F5.0)
    2 A = A + X
      A = A / 25.0
      PRINT 3, A
    3 FORMAT (7H1MEANΔ=, F10.4)
      STOP
      END
```

Note that the zeroing of A must be done before the loop begins, and the division of it by 25 and printing must be done after the loop is finished. No DIMENSION statement is needed because no array variables are used. Only one subject's score (X) is in memory at any time.

Program TWO. Compute a Mean, Any N

Now let us generalize the applicability of the program to any size sample. To do this we will assume that a control card is punched with the sample size in columns 1-5, and is added in front of the data deck.

```
      PROGRAM TWO (INPUT, OUTPUT)
      READ 1, N
    1 FORMAT (I5)
      A = 0.0
      DO 3 I = 1,N
      READ 2, X
    2 FORMAT (5X, F5.0)
    3 A = A + X
      A = A / N
      PRINT 4, A
    4 FORMAT (7H1MEANΔ=, F10.4)
      STOP
      END
```

Program THREE. Mean and Sigma

In the next routine we will compute sigma (parameter value, population standard deviation) as well as the mean of N scores.

```
      PROGRAM THREE (INPUT, OUTPUT)
      READ 1, N
    1 FORMAT (I5)
      A = 0.0
      S = 0.0
      DO 3 I = 1,N
      READ 2, X
    2 FORMAT (5X, F5.0)
      A = A + X
    3 S = S + X * X
```

```
      A = A / N
      S = (S / N - A * A)**0.5
      PRINT 4, A, S
    4 FORMAT (7H1MEANΔ=, F10.4 // 8HΔSIGMAΔ=, F10.4)
      STOP
      END
```

At the top of a new page, the output might look like this:

```
MEANΔ=ΔΔ235.1207

(blank line)

SIGMAΔ=ΔΔΔ22.4367
```

Program FOUR. Means, Sigmas, Correlation

In the next program we will extend the problem in two ways: We will expect two scores on each data card, compute means and sigmas for both variables, and also compute the correlation of the two variables. Formulas were given in Chapter II.

```
      PROGRAM FOUR (INPUT, OUTPUT)
      READ 1, N
    1 FORMAT (I5)
      AX = 0.0
      SX = 0.0
      AY = 0.0
      SY = 0.0
      R = 0.0
      DO 3 I = 1,N
      READ 2, X, Y
    2 FORMAT (5X, 2F5.0)
```

```
      AX = AX + X
      SX = SX + X * X
      AY = AY + Y
      SY = SY + Y * Y
    3 R = R + X * Y
      AX = AX / N
      AY = AY / N
      SX = (SX / N - AX**2)**0.5
      SY = (SY / N - AY**2)**0.5
      R = (R / N - AX * AY) / (SX * SY)
      PRINT 4, AX, AY, SX, SY, R
    4 FORMAT (8H1MEANSΔ=, 2F10.4 //
     1 9HΔSIGMASΔ=, 2F10.4 // 6HΔRXYΔ=, F10.4)
      STOP
      END
```

Program FIVE. Intercorrelations

Now we will extend the program to handle more than two variables, and we will have to change our strategy to use array variables, even though data from only one subject will be in memory at any given time. The program will be written to handle up to ten variables. It will compute means and sigmas for all of them, and will also compute a square matrix of correlations for all possible pairings of them.

We will extend our control card also by adding a field for the number of variables and fields to permit an alphameric heading to be read in and printed.

```
      PROGRAM FIVE (INPUT, OUTPUT)
      DIMENSION LAB (8), X(10), A(10), S(10), R(10,10)
```

```
C  READ CONTROL CARD.
      READ 1, NV, NS, LAB
    1 FORMAT (2I4, 8A9)
C  ZERO ACCUMULATORS.
      DO 2 I = 1,NV
      A(I) = 0.0
      S(I) = 0.0
      DO 2 J = 1,NV
    2 R(I,J) = 0.0
C READ DATA AND ACCUMULATE SUMS.
      DO 4 I = 1,NS
      READ 3, (X(J), J = 1,NV)
    3 FORMAT (5X, 10F5.0)
      DO 4 J = 1,NV
      A(J) = A(J) + X(J)
      S(J) = S(J) + X(J)**2
      DO 4 K = 1,NV
    4 R(J,K) = R(J,K) + X(J) * X(K)
C  COMPUTE MEANS AND SIGMAS.
      DO 5 I = 1,NV
      A(I) = A(I) / NS
    5 S(I) = (S(I) / NS - A(I)**2)**0.5
C  COMPUTE INTERCORRELATIONS.
      DO 6 I = 1,NV
      DO 6 J = 1,NV
    6 R(I,J) = (R(I,J) / NS - A(I) * A(J)) / (S(I) * S(J))
```

```
C  PRINT OUTPUT.
      PRINT 7, LAB, NV, NS
    7 FORMAT (1H1, 8A9  // I5, 10HΔVARIABLES //
     1 I5, 9HΔSUBJECTS)
      PRINT 8, (A(I), I = 1,NV)
    8 FORMAT ( / 9HΔΔMEANSΔ=, 10F10.4)
      PRINT 9, (S(I), I = 1,NV)
    9 FORMAT ( / 9HΔSIGMASΔ=, 10F10.4)
      PRINT 10
   10 FORMAT ( / 18HΔINTERCORRELATIONS)
      DO 11 I = 1,NV
   11 PRINT 12, (R(I,J), J = 1,NV)
   12 FORMAT ( / 9X, 10F10.4)
      STOP
      END
```

Exercises

1. How would you handle the possibility that one or more sigma values might be zero? (Division by zero is illegal; see statement six.)
2. How would you include labeling of the variables by number? Hint: This is a legal output variable list: (I, I = 1,NV)
3. How would you modify statement 11 to print only the lower triangular portion of the square correlation matrix, including the main diagonal?
4. Suppose a zero score means missing data. (A blank field is also read numerically as zero.) How would you detect subjects

with missing data, skip them in the accumulation phase and reduce NS?

5. Write a program to compute a two-group analysis of variance. Print the F-ratio and group means.

IX. Subroutines and Functions

There are two types of subprograms that can be called into operation by single statements in programs or other subprograms. The two types differ only in the manner in which they are called. Any operations, including I/O, can be carried out in a subprogram.

Because subprograms are independently compiled, variable names in them have no necessary connection with the same names in routines that call them, and statement numbers are also unique within subprograms and programs. The only linkage between a subprogram and the calling routine is through a list of arguments that appear in the calling statement and in the first statement of the subprogram. In the case of functions, the name of the function itself acts as a pseudo-variable; the purpose of the function routine is to define a value for it.

Subroutines

A subroutine is brought into operation by a CALL statement, such as the following:

```
CALL MEAN (A,X,5)
```

The first statement in the subroutine also has three arguments.

```
SUBROUTINE MEAN (AVG,DATA,N)
```

Arguments must correspond in number, order, and mode (integer or real). Note that constants and expressions may appear as arguments in CALL statements, but only variable names may appear as arguments in the subroutine. Arguments that are sent as constants must not be changed by operations within the subroutine.

In the example above, the memory location A in the calling routine is equated to that named AVG in the subroutine. X is similarly linked

to DATA, and N in the subroutine would hold the value five at the time of the CALL. Any array variable used as an argument must also appear in a DIMENSION statement in the subroutine. In some cases arrays may be dimensioned in the subroutine that are not used as arguments; usually this is "working space" needed only within the subroutine.

The program below obtains the mean of five values by calling subroutine MEAN to do the computation.

```
      PROGRAM AVERAG  (INPUT, OUTPUT)
      DIMENSION X(5)
      READ 1, X
    1 FORMAT (5F2.0)
      CALL MEAN (A, X, 5)
      PRINT 2, A
    2 FORMAT (7H1MEAN Δ=, F10.4)
      STOP
      END
```

When the program is submitted for execution, the deck for subroutine MEAN, shown below, would follow the program deck.

```
      SUBROUTINE MEAN (AVG, DATA, N)
      DIMENSION DATA (N)
      AVG = 0.0
      DO 1 I = 1,N
    1 AVG = AVG + DATA(I)
      AVG = AVG / N
      RETURN
      END
```

This example illustrates the fact that an integer variable may be used to specify array size in a subprogram, provided that both the array name and the size variable are arguments.

The RETURN statement is required at the point where control is to go back to the calling routine. Multiple RETURN statements may be used when the logic of the routine involves alternate paths.

Functions

The mode of the name of a subroutine is irrelevant, but the mode of a function name is important since it operates as a pseudo-variable in the calling statement. Although any operations can be carried out in a function, its primary purpose is to define a value for its name. Here is the function version of the previous example.

```
      FUNCTION AVG (DATA, N)
      DIMENSION DATA(N)
      AVG = 0.0
      DO 1 I = 1,N
    1 AVG = AVG + DATA(I)
      AVG = AVG / N
      RETURN
      END
```

Instead of a separate CALL statement, we include the function name with its arguments in an assignment statement.

```
      A = AVG(X, 5)
```

This function accomplishes the same thing as the previous subroutine, but returns the mean as the function name AVG, rather than as an argument.

Suppose we have another function named SIG which computes the sigma value and has the same arguments as AVG. Only one statement in a program would be needed to compute, for instance, a value two sigma units above the mean.

```
PLUS2 = AVG(X, 5) + 2.0 * SIG(X, 5)
```

When this statement is executed, the functions would be called before the multiplication and addition operations are carried out.

There would be no point in using a function or a subroutine in a program such as AVERAG where it was to be called only once. However, when a long and complex procedure can be logically divided into a few major operations, the programming task may be more manageable as a series of subroutines to be called by a "driver" program.

Fortran Functions

A wide variety of mathematical functions are automatically available to the Fortran compiler. We will discuss only a few of them here.

SQRT(X). The square root of the real-mode argument is returned. As is true of all subprogram arguments, X may be an expression or a constant as well as a variable name.

ABS(X) and IABS(K). The absolute (positive) value of the argument is returned. Separate real and integer mode versions are available.

MOD(KN,KD). The integer remainder produced by dividing the first argument by the second is returned. For instance, if KN is 5 and KD is 3, MOD would be returned as 2. This is useful in situations where one wishes to "unpack" a multi-digit number. Suppose we had a six-digit number in KX and wanted to identify the value represented by the middle two digits:

```
M2 = MOD(KX / 100, 100)
```

It is also useful for purposes such as determining whether a value is odd or even:

```
IF (MOD(KX, 2) .EQ. 0) GO TO 5
```

None of the Fortran functions alter the values sent to them as arguments.

In addition to the functions automatically available with the Fortran compiler, all academic computer installations make available to programmers one or more "libraries" of subprograms for various kinds of mathematical procedures, such as matrix multiplication.

Exercises

1. Write Function SIG so that it makes use of Function AVG.
2. Write Function COR (X, Y, N), which uses AVG and SIG to compute the correlation of variables X and Y over N cases.
3. Write a subroutine to be called instead of the PRINT statements and FORMATs 8, 9, and 12 in Program FIVE (page 45).
4. Write a subroutine that will "clean" a data vector of any length by changing values of 99 to zero.
5. Write a function that will return the factorial (N!) of its argument.

X. Some Useful Techniques

In this chapter we will present a few examples of common problems in social sciences data processing, and the programming strategies for dealing with them.

Temporary Storage

When data are input from cards, they can be read only once in a program. Many problems involve too many subjects to store all the data in central memory, yet require two or more passes through the data set. The solution is to write each subject's data to a temporary storage unit as it comes in from cards, and then read the data back from that unit later in the program.

One such problem is the computation of standard scores, which requires initial computation of means and sigmas. A common standard score scale has a mean of 50 and sigma of 10. The formula is:

$$Z = (X - \mu) / \sigma * 10 + 50$$

In the program below we assume a sample of 1000 cases and 15 variables, each to be separately standardized. Unit 4 will be used for temporary storage of the data file. Output would consist of 1000 punch cards containing the standardized scores.

```
      PROGRAM ZSCORE (INPUT, PUNCH, TAPE 4)
      DIMENSION X(15), A(15), S(15)
      REWIND 4
      DO 1 I = 1,15
      A(I) = 0.0
    1 S(I) = 0.0
      DO 4 I = 1,1000
      READ 2, ID, X
```

```
2 FORMAT (A4, 6X, 15F3.0)
  DO 3 J = 1,15
  A(J) = A(J) + X(J)
3 S(J) = S(J) + X(J)**2
4 WRITE (4,2) ID, X
  DO 5 I = 1,15
  A(I) = A(I) / 1000.0
5 S(I) = SQRT(S(I) / 1000.0 - A(I)**2)
  REWIND 4
  DO 7 I = 1,1000
  READ (4,2) ID, X
  DO 6 J = 1,15
6 X(J) = (X(J) - A(J)) / S(J) * 10.0 + 50.0
7 PUNCH 2, ID, X
  STOP
  END
```

Key Cards

Scoring tests or questionnaires is a very common data-processing chore. The strategy exemplified here is the use of a special key card preceding the data deck. If the problem was one of scoring a multiple-choice test, this card would contain the right-answer choice number for each item. If the problem involved adding up item scores of subsets of items to form scales, this card would hold the scale number to which each item is assigned. In either case, each column of the key card represents the key code for one variable, with column one for variable one, column two for variable two, etc.

The example program below concerns computation of five scale scores, ten items each, randomly ordered on the questionnaire. Each column of the key card through column 50 will hold a scale number between one and five. We will assume 1000 questionnaires to be scored, with punched output of scale scores.

```
      PROGRAM SCALE (INPUT, PUNCH)
      DIMENSION KEY(50), X(50), S(5)
      READ 1, KEY
    1 FORMAT (50I1)
      DO 5 I = 1,1000
      READ 2, ID, X
    2 FORMAT (A4, 6X, 50F1.0)
      DO 3 J = 1,5
    3 S(J) = 0.0
      DO 4 J = 1,50
      K = KEY(J)
    4 S(K) = S(K) + X(J)
    5 PUNCH 6, ID, S
    6 FORMAT (A4, 6X, 5F3.0)
      STOP
      END
```

Sorting

It is not at all uncommon to encounter the need to sort a list of scores or subject code numbers in the process of programming a procedure. In this case we will assume that the data set is small enough to be held in memory. When a tape file must be sorted, the programmer normally uses

one of the standard routines available at most installations.

How might you approach this problem? The simplest approach, called a "bubble sort" because the low values gradually float to the top of the list as the high ones sink, is to pass through the list, comparing adjacent values, and swapping any pair out of order. This is repeated until no swaps occur on a complete pass.

Somewhat faster is the "delayed swap" method which involves scanning the rest of the list on each pass to locate the lowest value remaining, and only then swapping with the one indexed by the pass number.

A still faster method, especially with long arrays, called the Shell-Metzner Sort,* is not nearly so intuitively appealing, but it capitalizes on the fact that if A < B and B < C, then there is no reason to compare A and C. This method is programmed below in the form of a subroutine, which assumes the data to be stored in a subjects (rows)-by-variables (columns) matrix. Sorting takes place on the values of the first variable, with the other variables (if any) carried along on each swap.

Arguments to this subroutine are:

X The name of the data matrix

N The number of rows of X, cases to be sorted

KV The number of columns of X, variables

*Grillo, J.P. A comparison of sorts, Creative Computing, Nov.-Dec., 1976, 76-81.

```
      SUBRCUTINE SORT (X, N, KV)
      DIMENSION X(N,KV)
      M = N
    1 M = M / 2
      IF (M .EQ. 0) RETURN
      K = N - M
      J = 1
    2 I = J
    3 L = I + M
      IF (X(I,1) .LT. X(L,1)) GO TO 5
      DO 4 JV = 1,KV
      T = X(I,JV)
      X(I,JV) = X(L,JV)
    4 X(L,JV) = T
      I = I - M
      IF (I .GE. 1) GO TO 3
    5 J = J + 1
      IF (J .GT. K) GO TO 1
      GO TO 2
      END
```

Buffered Aggregation

Particularly in educational research, data often are collected from individuals, but must be aggregated to a higher unit such as the classroom before statistical analyses can be carried out. Means of the

individual scores are most often used as the aggregate measures. Data processing of this kind is usually complicated by the fact that the classes differ in size, and the use of "header" cards is not convenient.

In the following example we will assume that data from 1000 students in 30 classes are stored on magnetic tape (unit 2), and that they have been sorted by a class code number in columns 1-2. Each record contains five variables for which class means are to be computed and punched.

Each record is read from the tape to a "buffer" array (X). If it is from the same class as the previous record, it is added to a "class" array (C). (Classes are known to contain no more than 40 students.) If it is <u>not</u> from the same class, and is, therefore, the first record of a new class, a subroutine is called to compute and punch means from the data in C. Then X is shifted to C to begin a new class.

```
      PROGRAM CLASS (PUNCH, TAPE 2)
      DIMENSION X(6), C(6,40)
      REWIND 2
C  INITIALIZE CLASS ARRAY.
      READ (2,1) X
    1 FORMAT (F2.0, 3X, 5F3.0)
      N = 1
      DO 2 I = 1,6
    2 C(I,N) = X(I)
C  READ REMAINING RECORDS.
      DO 6 I = 1,999
      READ (2,1) X
      IF (X(1) .NE. C(1,1)) GO TO 5
```

```
      N = N + 1
    3 DO 4 I = 1,6
    4 C(I,N) = X(I)
      GO TO 6
    5 CALL GROUP (N, C)
      N = 1
      GO TO 3
    6 CONTINUE
C   OUTPUT LAST CLASS.
      CALL GROUP (N, C)
      STOP
      END

      SUBROUTINE GROUP (N, C)
      DIMENSION C(6,40), A(5)
      DO 1 I = 1,5
    1 A(I) = 0.0
      DO 2 I = 1,N
      DO 2 J = 1,5
    2 A(J) = A(J) + C(J+1,I)
      DO 3 I = 1,5
    3 A(I) = A(I) / N
      PUNCH 4, C(1,1), A
    4 FORMAT (F2.0, 3X, 5F10.4)
      RETURN
      END
```

60

Exercises

1. Write a subroutine that does what SORT does, but by means of the bubble-sort method.
2. Write a program to compute the odd-even reliability of any set of up to 100 items.
3. Write a program that will ipsatize a data set by converting each subject's set of scores to percentages of his total.
4. Write a program that will locate all pairs of subjects in a sample of 1000 who have identical sets of five scores.
5. Modify Program CLASS to use Subroutine SORT in case the input file is not already sorted by class code.

Appendix A: Differences Between CDC and IBM Fortran

There are a number of differences between these two "dialects" in addition to those listed below, but they are not relevant to the level of programming sophistication for which this manual is intended.

Word Size and Character Codes

IBM machines of the 360/370 series have 32-bit words of memory, and an 8-bit coding system is employed for alphameric storage. Therefore, a maximum of four characters may be stored in one word of memory.

Program Name Card

IBM Fortran does not use a card of this type. Equipment buffers are designated with system control cards. If you want to name your program anyway, use a comment card as the first in the deck.

Appendix B: Summary of Fortran Statements

DIMENSION Reserves memory areas for array variables.

Example: DIMENSION A(3), B(4,2), C(3,2,5)

Assignment: Changes content of location on the left of the equals sign.

Example: A(K) = 2.0 * B(I,K) + 8.0

GO TO Unconditional transfer to a numbered statement.

Example: GO TO 12

IF Conditional execution of a statement if proposition in parentheses is true.

Example: IF (A .EQ. B .AND. B .NE. C) K = K + 1

DO Initializes a loop.

Example: DO 15 I = 1,10,2

CONTINUE Does nothing but carry a statement number.

Example: 25 CONTINUE

READ Input data from punch cards.

Example: READ 22, A, (B(I), I = 1,20)

PRINT Output to a line printer.

Example: PRINT 24, KB, D

PUNCH Output on punch cards.

Example: PUNCH 3, ID, X

REWIND Position mass storage file at start point.

Example: REWIND 2

READ Input of one or more records from mass storage.

Example: READ (2,15) A, (B(I), I = 1,10)

WRITE Output of one or more records to mass storage.

Example: WRITE (3,22) ((C(I,J), J = 1,2), I = 1,3)

FORMAT Describes fields of records on an external medium. Referenced by I/O statements.

Example: 22 FORMAT (A3, 2X, 2I3, 3F5.3)

CALL Initiates operation of a subroutine. Linked only through argument list.

Example: CALL COR (NV, NS, R, 10)

SUBROUTINE First statement of a subroutine.

Example: SUBROUTINE COR (N, K, R, J)

FUNCTION First statement of a function routine. Linkage through name and arguments.

Example: FUNCTION AVG (N, X)

Function Call By appearance in an expression.

Example: A = AVG (10, D) + SQRT (X(2))

RETURN Transfer control back from subroutines and functions.

STOP Termination of program execution.

END Last statement in every program, function, and subroutine.